IN THE SPECIFICATION

Please replace the paragraph starting on page 1, line 4 with the following paragraph:

The present application may relate to co-pending ~~application~~ applications Serial No. ~~_____, filed concurrently (Docket No. 0325.00407)~~ 09/676,704, filed September 29, 2000, Serial No. ~~_____, filed concurrently (Docket No. 0325.00408)~~ 09/676,171, filed September 29, 2000, now U.S. Patent No. 6,578,118, issued June 10, 2003, Serial No. ~~_____, filed concurrently (Docket No. 0325.00410)~~ 09/676,705, filed September 29, 2000, now U.S. Patent No. 6,631,455, issued October 7, 2003, Serial No. ~~_____, filed concurrently (Docket No. 0325.00411)~~ 09/676,170, filed September 29, 2000, now U.S. Patent No. 6,581,144, issued June 17, 2003, Serial No. ~~_____, filed concurrently (Docket No. 0325.00433)~~ 09/676,169, filed September 29, 2000, now U.S. Patent No. 6,816,955, issued November 9, 2004, which are each hereby incorporated by reference in their entirety.

Please replace the paragraph beginning at page 7, line 8 with the following paragraph:

Another aspect of the present invention concerns an apparatus for initializing a default value of a queue. The apparatus ~~comprising~~ may comprise a memory section having a first

storage element and a second storage element. The apparatus may be configured to pass the default value of the queue and initialize the default value of the queue without writing to the memory section.

Please replace the paragraph beginning at page 8, line 1 with the following paragraph:

The objects, features and advantages of the present invention include providing a method and/or architecture for implementing a virtual multiqueue FIFO that may (i) be implemented with efficient use of memory storage, (ii) implement a single memory device or package, (iii) allow flexibility for implementing variable sized FIFOs, (iv) allow increments/decrements (e.g., changing) of maximum depth of the individual FIFOs, (v) allow depth ~~increments/de-increments~~ increments/decrements in small quantities, (vi) allow scalability for a number of FIFOs, (vii) ~~create~~ provide a virtual multiqueue FIFO in a single device, (viii) minimize initialization or configuration time of ~~the~~ a system before packet transfer can be started, (ix) allow multiplexing without any overhead and/or (x) implement a single port memory to implement a virtual multiqueue FIFO with a number of clock domains, where the virtual multiqueue FIFO logic generally operates at a fastest clock rate of a plurality of clock rates.

Please replace the paragraph beginning at page 10, line 11 with the following paragraph:

FIG. 16 is a ~~flow~~ block diagram of an initialization circuit in accordance with the present invention;

Please replace the paragraph beginning at page 10, line 15 with the following paragraph:

FIG. 18 is a block diagram of an arbitration logic block of the present invention;

Please replace the paragraph beginning at page 11, line 6 with the following paragraph:

Referring to FIG. 2, a block diagram of a system 100 is shown in accordance with a preferred embodiment of the present invention. The system 100 may efficiently implement a number of virtual multiqueue FIFOs 103a-103n, where n is an integer. The system 100 may implement a single memory 102 and create one or more virtual multiqueue FIFOs 103a-103n in the single memory 102. The system 100 may allow flexibility for implementing variable sized FIFOs 103a-103n. For example, the system 100 may allow changing of

depth of the FIFOs 103a-103n. Such depth increments may be made in small quantities (e.g., as small as one memory block, where a block may include a number of locations, such as 8 locations) and may be scalable for a number of FIFOs. However, a particular number of blocks and/or locations may be varied in order to meet the criteria of a particular implementation. Additionally, the system 100 may allow an increased size of a particular FIFO 103a-103n to ~~impacting~~ not impact the total number of FIFOs 103a-103n. The system 100 may allow the single port memory 102 to implement virtual multiqueue FIFOs 103a-103n with a number of clock domains, where the virtual multiqueue FIFOs 103a-103n and associated logic may operate at a preferred (e.g., the fastest) clock rate.

Please replace the paragraph beginning at page 14, line 6 with the following paragraph:

The memory 102 generally has ~~only~~ one port for read and write operations of the memory 102. ~~Thus the~~ The memory 102 may have a width that may be twice the width of the read/write port. For example, if the memory 102 has a read port x40 and a write port x40, the memory 102 has only one port and cannot perform two operations simultaneously (with both the read and write frequencies about the same and the main memory is running at the same frequency). In order to allow simultaneous read/write operations,

the memory 102 may be twice as wide and may retain a high frequency of operation. The system 100 may allow an external device (not shown) to receive an uninterrupted data stream (via the signal DATA_OUT), since the single port memory 102 and the logic 108 may operate in the clock domain FAST_CLOCK_DOMAIN.

Please replace the paragraph beginning at page 17, line 4 with the following paragraph:

Referring to FIG. 4, a detailed diagram of the address generation block 122 is shown. The address generation block 122 generally comprises an address logic block (or circuit) 130, a storage element (or circuit) 132 and a storage element (or circuit) 134. In one example, the address logic block 130 may be implemented as a FIFO address to physical address logic block, the storage element 132 may be implemented as a forward pointer memory and the storage element 134 may be implemented as a FIFO pointer memory. However, ~~the~~ each of the circuits 130, 132 and/or 134 may be implemented as another appropriate type device in order to meet the criteria of a particular implementation.

Please replace the paragraph beginning at page 18, line 18 with the following paragraph:

The forward pointer location may determine a next (e.g., start) address location. The FIFO pointer location may determine both a start and an end address location. The memory 134 may be implemented to store a FIFO pointer location. The address logic 130 may convert a FIFO address to a physical memory address (e.g., PHY_ADDR). The address logic 130 may create the virtual FIFOs 103a-103n by maintaining link lists ~~into~~ in the memory 132. The address logic 130 generally stores start and end pointer information for each FIFO ~~into~~ in the memory 134. When a new queue address is requested for a read or write operation, the address logic block 130 generally requests the data from the pointer memory 134. When the read or write operation for the queue is completed and a new queue address is requested, the previous queue data is stored back in the FIFO pointer memory 134.

Please replace the paragraph beginning at page 19, line 18 with the following paragraph:

The write interface 104' generally receives data via the signal DATA_IN. The write interface 104' may communicate with the logic block 108' by presenting/receiving a signal (e.g., WR_CTRL) to determine an address for storing the incoming data and status of the flags for a particular queue. The write interface 104' may then write the data into the dual port memory 102'. The read

interface 106′ may ~~then~~ receive a read address and flag status of the FIFOs from the logic block 108′ (via the signal RD_CTRL) and may read the data from the dual port memory 102′.

Please replace the paragraph beginning at page 20, line 6 with the following two paragraphs:

The logic block 108′ may (i) synchronize the data from the two clock domains (e.g., the clock domains WR_CLK_DOMAIN and RD_CLK_DOMAIN), (ii) arbitrate the access to the dual port memory 102′ from the write interface 104′ and the read interface 106′, (iii) compute arbitration flags and (iv) if the FIFO is not full, provide a respective address for the dual port memory ~~102′. The~~ 102′.

The system 100 (or 100′) may implement a number of virtual multiqueue FIFOs in a single memory. The virtual multiqueue FIFO implementation 100 may be flexible for implementing variable sized FIFOs, since the size of each FIFO may be changed independently. Additionally, depth ~~increments/de-increments~~ increments/decrements of the virtual multiqueue FIFOs may be made in small quantities. The virtual multiqueue FIFOs may allow an increased size of one FIFO 103a-103n to not impact the total number of FIFOs 103a-103n. Furthermore, the virtual multiqueue FIFOs 103a-103n may be scalable for implementing a large number of FIFOs.

Please replace the paragraph beginning at page 21, line 1 with the following paragraph:

The system 100 may create a number of virtual FIFOs 103a-103n in a single device. The system 100 may implement a single port memory to implement a virtual multiqueue FIFO with two clock domains, where the virtual multiqueue FIFO logic operates at the preferred (faster) clock rate. The system 100 may allow virtual multiqueue FIFOs 103a-103n to implement a single storage element for data. The system 100 may provide a control logic (e.g., the logic block 108) for constructing the virtual FIFOs 103a-103n in the storage element. The system 100 may provide a read and a write interface block for synchronizing data and control signals. Additionally, the system 100 may provide a fast clock domain for configuring the control logic (e.g., the control arbitration flag generation 108 and a storage element 102, clocked in the clock domain FAST_CLOCK_DOMAIN).

Please replace the paragraph beginning at page 22, line 14 with the following paragraph:

The controller 202 may generate (i) a number of control signals (e.g., ADD_REQ_INF, SKIP_ADDR_REQ and LOOK_AHEAD_ADDR_REQ)

that may be presented to the address generator block 204, (ii) a control signal (e.g., LOGIC_CTRL) that may be presented to the logic block 208 and (iii) a signal (e.g., CTRL) that may be presented to the write interface 206. The address generator 204 may generate one or more ~~signal(s)~~ signals (e.g., ADDR) that may be presented to the memory 210. The signal ADDR may specify pointer locations based on the signals ADD_REQ_INF, SKIP_ADD_REQ and LOOK_AHEAD_ADDR_REQ. The ~~memory addresses~~ signal ADDR may be implemented comprising memory addresses for the memory block 210, based on requests (e.g., the signals ADDR_REQ_INF, SKIP_ADDR_REQ and LOOK_AHEAD_ADDR_REQ) from the controller 202.

Please replace the paragraph beginning at page 23, line 5 with the following paragraph:

When the controller 202 issues the signal SKIP_ADDR_REQ to the address generator 204, the address corresponding to the port information for the current packet is generally skipped. The address generator 204 may provide an address for a first location after the port information location. The configurations of the controller 202 and the address generator 204 may allow the system 200 to internally steal cycles by skipping the port information locations. The stolen cycles may be ~~then~~ used at an end of packet (EOP) to read the port information location for the next packet.

Please replace the paragraph beginning at page 23, line 14 with the following paragraph:

The controller 202 may then issue a command (e.g., the signal LOOK_AHEAD_ADDR_REQ), requesting an address for the port information from a next packet. The command LOOK_AHEAD_ADDR_REQ may have a jump value that may be predetermined at power-up. However, the command LOOK_AHEAD_ADDR_REQ may be alternately configured in order to meet the criteria of a particular implementation. The command LOOK_AHEAD_ADDR_REQ may allow the port information register 208 to be updated with the information from the next packet by the time the EOP data is output.

Please replace the paragraph beginning at page 29, line 10 with the following paragraph:

The logic block 400 generally comprises an address generator block (or circuit) 402, a logic block (or circuit) 404 and a logic block (or circuit) 406. The address generator circuit 402 may be implemented as a ~~pointer to~~ memory address pointer generator circuit. The logic block 404 may be implemented as a head pointer logic circuit. The logic block 406 may be implemented as a multicast head pointer logic circuit. However, particular

implementations of the blocks 402, 404 and 406 may vary in order to meet the design criteria of a particular implementation.

Please replace the paragraph beginning at page 31, line 8 with the following paragraph:

The address generator circuit 402 may (i) generate ~~(i)~~ a next address for the same queue (in the case of unicast queue) or (ii) hold data without a pointer pop for the same multicast queue. The address generator 402 may generate an appropriate address via the signal MEMORY_ADDRESS. The signal NEWQ_HPTR may be used by the address generator 402 to generate the address MEMORY_ADDRESS when a switched queue condition occurs. The signal ~~POPED_HPTR~~ POPPED_HPTR may be used when (i) data from a current block is completely read and (ii) data from a next block is to be read. The signal MULTICAST_HPTR is generally implemented when a next read is from the same multicast queue. Additionally, the signal FIRST_MC_HPTR may be implemented for a first block for the multicast queue and may be directly loaded from a write side.

Please replace the paragraph beginning at page 33, line 4 with the following paragraph:

The logic block 406 generally comprises a demultiplexer block (or circuit) 440, a number of logic blocks 442a-442n (where n is an integer), a multiplexer 444, and a logic block (or circuit) 446. The ~~logic~~ demultiplexer circuit 440 may be implemented as a (4-n) demultiplexer, where n may represent the number of multicast queues supported. The registers 442a-442n may each be implemented as a multicast head pointer storage logic circuits. The multiplexer 444 may be implemented as an (n-1) multiplexer. The logic 446 may be implemented as a multicast head pointer flush storage logic circuit. In one example, the logic ~~circuit 430~~ circuits 442a-442n may be implemented as a register. In another example, the logic ~~circuit 430~~ circuits 442a-442n may be implemented as a latch. However, the logic ~~circuit 430~~ circuits 442a-442n may be implemented as another appropriate type device in order to meet the criteria of a particular implementation. The demultiplexer 440, the logic blocks 442a-442n, the multiplexer 444 and the logic block 446 may be implemented for a multicast queue operations.

Please replace the paragraph beginning at page 36, line 20 with the following paragraph:

Referring to FIG. 12, a flow diagram 450 is shown illustrating data flow during a unicast read operation of the

system 400. The flow diagram 450 may illustrate a detailed and simplified implementation of the circuit 400 during a unicast read operation. The flow diagram 450 generally comprises a queue pointer select state 420, a memory address generation logic state 422, a next memory address generation logic state 424 and a head pointer register logic state 430. The states 420, 422, 424 and 430 generally correspond to the circuit elements of FIG. 11. The queue pointer select state 420 generally presents a signal to the memory address generation logic state 422 in response to the signals POPPED_HPTR, ~~the signal~~ NEWQ_HPTR and ~~the signal~~ UNICAST_HPTR. The memory address generation logic state 422 generally presents the memory address signal MEMORY_ADDRESS in response to the signal received from the queue pointer select state 420. The memory address generation logic 422 also presents a signal to the next memory address generation logic 424. The next generation logic state 424 presents a signal to the head pointer register ~~generation~~ logic 430. The head pointer register ~~generation~~ logic 430 generates the signal UNICAST_HPTR in response to a feedback ~~in~~ of the signal UNICAST_HPTR and the signal received from the next memory address generation logic 424.

Please replace the paragraph beginning at page 41, line 3 with the following paragraph:

The multiplexer 510 is generally configured to receive an output of the default depth value register 506 and the configurable depth register 508c. A depth output of the multiplexer 510 is generally controlled by the queue depth score-card logic ~~506~~ 507. The multiplexer 510 may select the depth value for the selected queue in response to the queue depth score-card logic 507.

Please replace the paragraph beginning at page 41, line 19 with the following paragraph:

The logic block 500 may allow a customer to initialize a configurable depth field of a queue without writing to the memory. The present invention generally comprises ~~of~~ a programmable queue pointer memory (504), a default depth value register (506), a queue configuration status storage element (507) and a multiplexer (510) for passing an appropriate queue depth value. The programmable default value (or hardwired default) may minimize the default depth and allow for change of depth as needed.

Please replace the paragraph beginning at page 49, line 1 with the following paragraph:

The logic block 902 may have a number of inputs 910a-910n, where n is an integer. The input 910a may receive a signal

(e.g., SYSTEM_CLK). The input 910b may receive one or more ~~signal(s)~~ signals (e.g., WRITE_DATA). The input 910c may receive one or more signal(s) (e.g., WRITE_ADD). The input 910n may receive one or more signal(s) (e.g., WRITE_ENB). The logic block 902 may have an output 912 and an output 914. The output 912 may present a signal (e.g., WRITE_DATA_DP). The signal ~~WRITE_DATA_DP)~~ WRITE_DATA_DP may be presented both to the memory 906 and to the multiplexer 908. The output 914 may present a signal (e.g., WR) to the memory 906. The signal WR may comprise an address signal (e.g., WRITE_ADD_DP) and an enable signal (e.g., WRITE_ENB_DP). The logic circuit 902 may also have an output 916 that may present a signal (e.g., WRITE_ADD_SYS) and an output 918 that may present a signal (e.g., WRITE_ENB_SYS).

Please replace the paragraph beginning at page 50, line 1 with the following paragraph:

The logic circuit 904 may have a number of inputs 930a-930n, where n is an integer. The input 930a may receive the signal READ_ENB_SYS. The input 930b may receive the signal READ_ADD_SYS. The input 930c may receive the signal WRITE_ENB_SYS. The input 930d may receive the signal WRITE_ADD_SYS. The input 930n may receive the signal SYSTEM_CLK. The logic block 904 may have an output 928 that may present a signal (e.g., MUX_SEL) to a control

input of the multiplexer 908. The signal MUX_SEL generally selects either ~~the~~ a signal from an output of the dual port memory 906 or the signal WRITE_DATA_DP to present at an output of the circuit 900. The multiplexer ~~920~~ 908 may be configured in response to the signals READ_ENB_SYS, READ_ADD_SYS, WRITE_ENB_SYS, WRITE_ADD_SYS and/or SYSTEM_CLK (via the logic circuit 904).

Please replace the paragraph beginning at page 51, line 7 with the following paragraph:

The logic circuit 904 generally compares the signals READ_ADD_SYS and WRITE_ADD_SYS. If the address signals (e.g., READ_ADD_SYS and WRITE_ADD_SYS) match and if the enable signals (e.g., WRITE_ENB_SYS and READ_ENB_SYS) are active, the logic circuit 904 may indicate contention. When a contention occurs, the data (e.g., the signal WRITE_DATA_DP) from a write register (e.g., a register block 954 to be discussed in connection with FIG. 20) may be directly passed (through the multiplexer 908) to ~~the read data interface 905~~ the output of the circuit 900 while the data is being written into the dual-port memory 906.

Please replace the paragraph beginning at page 51, line 16 with the following paragraph:

Referring to FIG. 20, an exemplary implementation of the write interface logic 902 is shown. The write interface logic 902 generally comprises a synchronizer block (or circuit) 950, a multiplexer block (or circuit) 952 and ~~a~~ the register block (or circuit) 954 ~~circuit 954)~~. The synchronizer circuit 950 generally receives the signals SYSTEM_CLK, WRITE_DATA, WRITE_ENB and WRITE_ADD. The multiplexer 952 may also receive the signals WRITE_DATA, WRITE_ENB and WRITE_ADD. The multiplexer 952 may also receive a signal from the ~~synchronizers~~ synchronizer block 950. In the case when the system clock SYSTEM_CLK and the write clock WRITE_CLK are not the same, the data is first synchronized and then passed to the system output. Otherwise the data, address and enable signals are directly passed to the system interface.

Please replace the paragraph beginning at page 52, line 15 with the following paragraph:

The signal SYSTEM_CLK may be selected from either the read clock READ_CLK or the write clock WRITE_CLK based on a speed of the respective clocks. The write interface logic 902 may generate the signals WRITE_DATA_SYS, WRITE_ADD_SYS and WRITE_ENB_SYS, synchronized with the system clock SYSTEM_CLK. The write interface logic block 902 may also generate the signals WRITE_DATA_DP, WRITE_ADD_DP and WRITE_ENB_DP for writing to the

dual port memory 906. Similarly, the read logic block 905 may generate the signals READ_ADD_SYS, READ_ENB_SYS, ~~and the~~ READ_ADD_DP and READ_ENB_DP. The read/write contention logic block 904 generally looks at the read and write signals one clock cycle ahead of a write operation (e.g., before data is written to the dual-port memory 906). In the case of a read/write contention, the write may have priority and the data from the write register 954 (e.g., the signal WRITE_DATA_DP) may pass to the output of the circuit 900 through the multiplexer ~~920~~ 908 while being written to the dual port memory 906. The circuit 900 may not extend a write cycle because of contention between ~~read/write~~ simultaneous read/write operations.